Jeffrey Jason Stroth

14 Nov 2017

Data Structures

Prof. Woodcock

Hash Table Report

The hash values computed for project six derived from the given text file containing paragraphs of strings. I used a word count to factor how many words were contained in the text file. Once this number was found I doubled to bring down the number of collisions that may occur. I then used a readText function to pull a string one text at a time that then sent the data value to a function that would add to the hash table. Using the netbeans built in hash function, I added each data value to my hash table class.

What I discovered is that the hash table runs the same each time and while it randomly chose where to put the words, they are placed in the same spot of the array every time. I also discovered that the closer the array size is to the amount of data being entered, the greater the amount of collisions there will be. Another thing I discovered, is that if you don’t write code to handle a collision, the program will drop the data from being input and move on, even if the data values aren’t identical.

There are a few key differences when comparing the binary tree, the 234 tree and a hash table. The biggest difference is sorting. If you have data that needs to be in alphabetical or numerical order, you will need to use the two trees because the hash table randomly assigns a location for data. Another difference is size. To avoid collisions, a hash tables array needs to be much larger than the data needed. Compared to the other two that can fill the array and gradually increase size when needed. I think the binary tree is better. Even though a hash table has a search potential of O(1), there is a lot that goes into rehashing and collision detection to maintain this. This is compared to the binary search of O(Log2N) that isn’t as fast, but can be sorted and is much more simple.